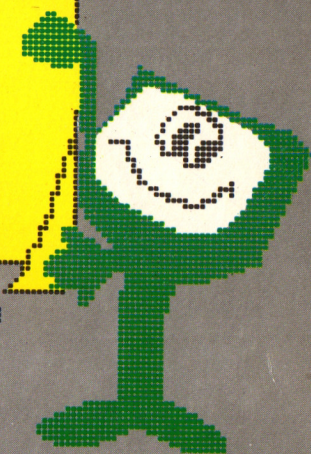
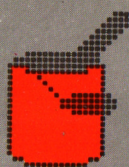
20 VIDEOLEZIONI DI BASIC
PER IMPARARE CON L'MSX

La CPU del MSX

I BUS, BIT e BYTE

Sistema binario
ed esadecimale

Tecniche di correzione
e chiarezza

REM-GOTO-IF THEN
ELSE-CLS

Videoesercizi

Videogioco n. 3

VIDEOBASIC MSX
Pubblicazione quattordicinale
edita dal Gruppo Editoriale Jackson
Direttore Responsabile:
Giampietro Zanga
Direttore e Coordinatore
Editoriale: Roberto Pancaldi
Autore: Softidea -
Via Indipendenza 88-90 - Como
Redazione software:
Alessandro Brunetti
Francesco Franceschini,
Luciano Magrini
Progetto grafico:
Studio Nuovidea - via Longhi, 16 - Milano
Impaginazione:
Moreno Confalone
Illustrazioni:
Cinzia Ferrari, Silvano Scolari
Fotografie:
Marcello Longhini
Distribuzione: SODIP
Via Zuretti, 12 - Milano
Fotocomposizione: Lineacomp S.r.l.
Via Rosellini, 12 - Milano
Stampa: Grafika '78
Via Trieste, 20 - Pioltello (MI)
Direzione e Redazione:
Via Rosellini, 12 - 20124 Milano
Tel. 02/6880951/5

Autorizzazione alla pubblicazione Tribunale di Milano n° 422 del 22-9-1984
Spedizione in abbonamento postale Gruppo II/70 (autorizzazione della Direzione Provinciale delle PPTT di Milano).
Prezzo del fascicolo L. 8.000
Abbonamento comprensivo di 5 raccoglitori L. 165.000
I versamenti vanno indirizzati a: Gruppo Editoriale Jackson S.r.l. - Via Rosellini, 12 20124 Milano, mediante emissione di assegno bancario o cartolina vaglia oppure utilizzando il c.c.p. n° 11666203.
I numeri arretrati possono essere richiesti direttamente all'editore inviando L. 10.000 cdu. mediante assegno bancario o vaglia postale o francobolli.
Non vengono effettuate spedizioni contrassegno.

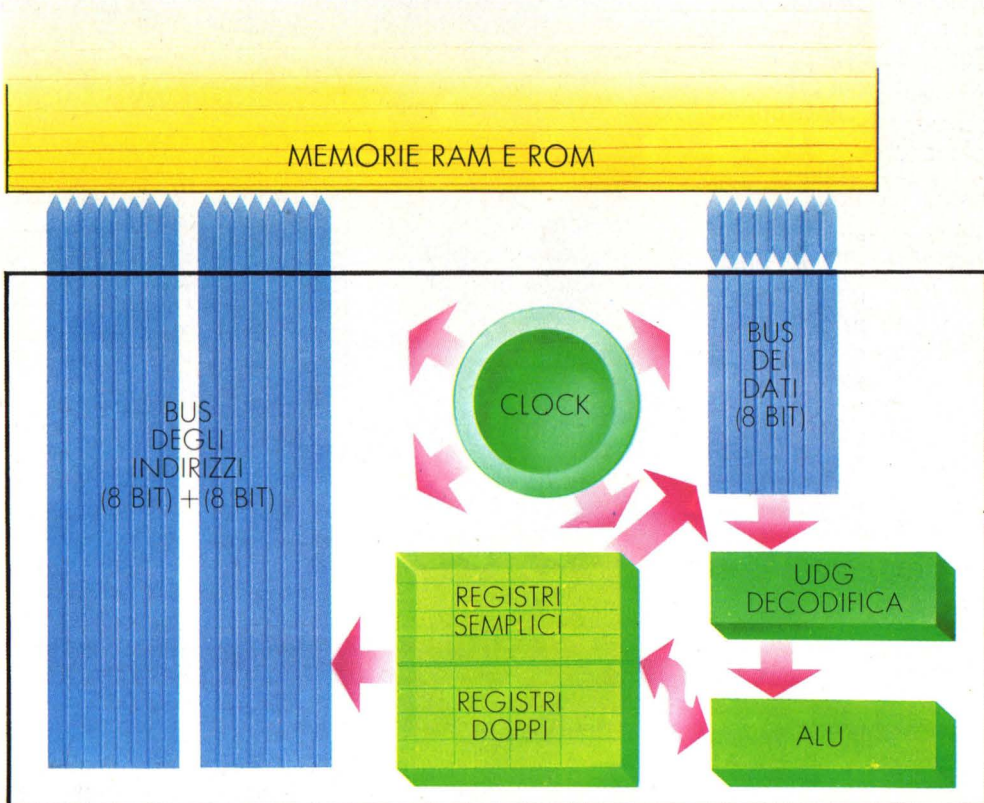
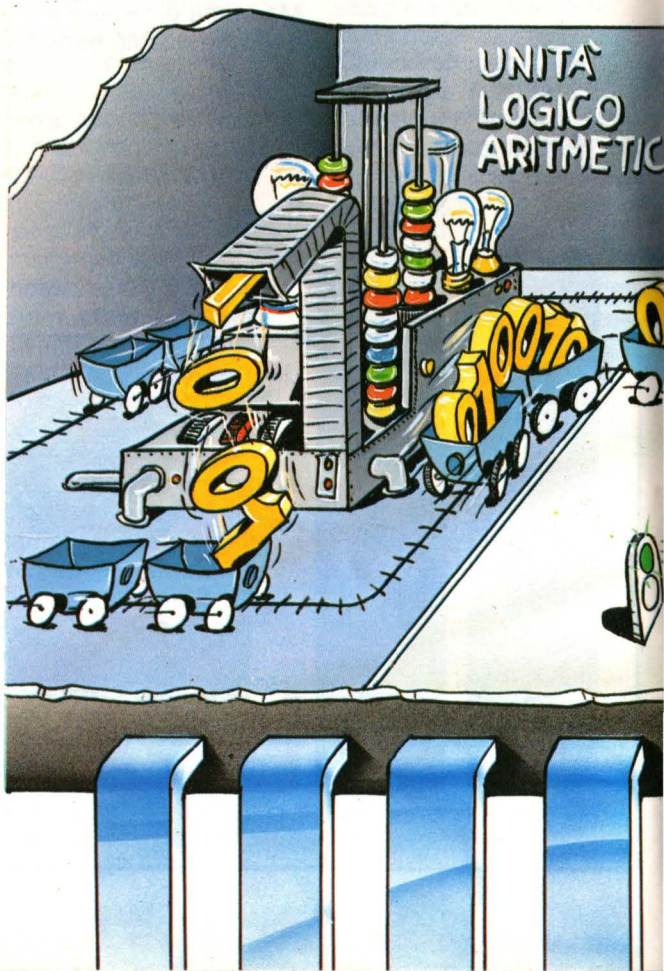
SOMMARIO

Introduzione

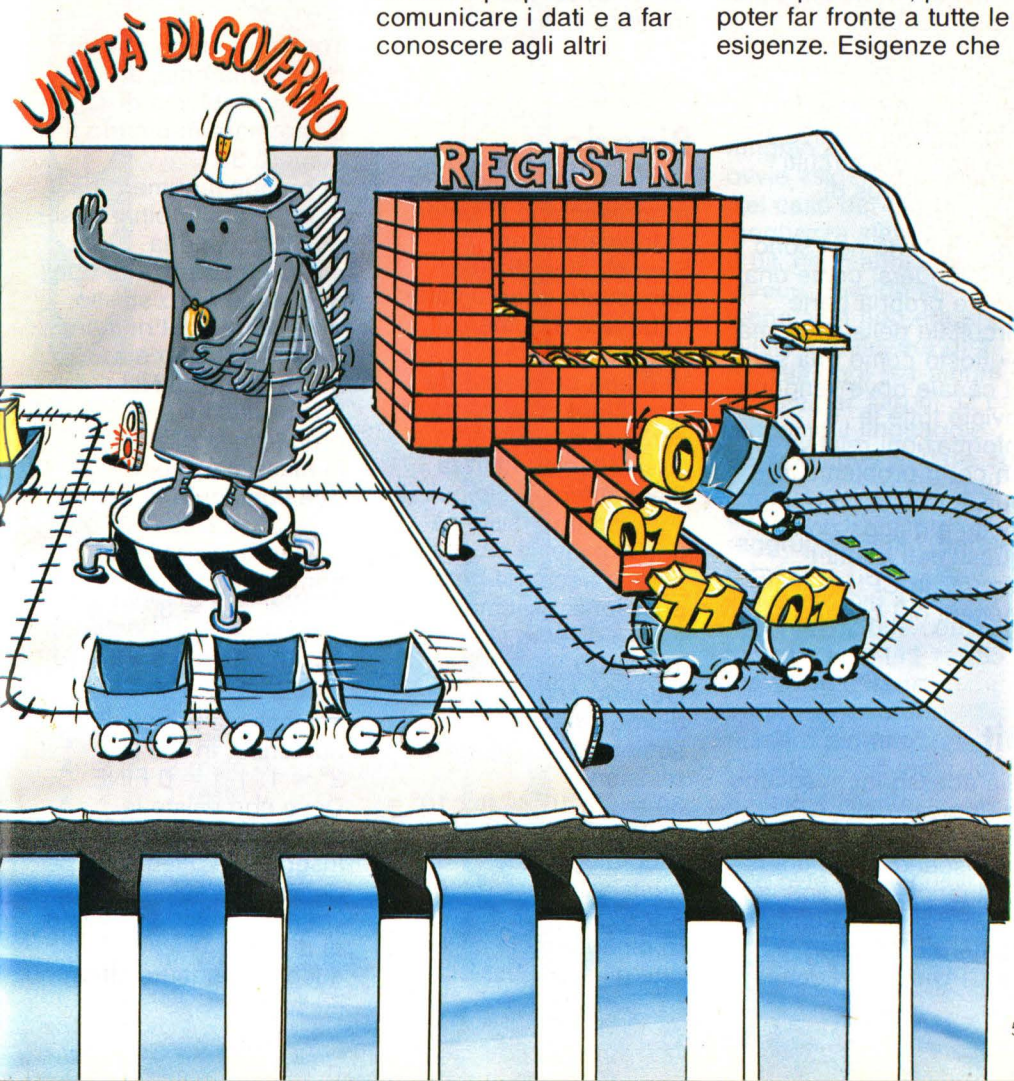
Nella prima parte, come vedrai, parleremo dell'unità fondamentale di qualsiasi elaboratore, la CPU, spiegando qual'è la sua funzione; gli strani codici che adopera ci indurranno ad esaminare i numeri binari ed esadecimali.
Affronteremo quindi il problema di come scrivere bene i programmi parlando di EDITOR e di come ricercare e correggere gli errori (DEBUG). Infine, dopo aver appreso il funzionamento dell'istruzione di salto condizionato IF THEN GOTO, potremo finalmente far eseguire al nostro computer dei compiti di una certa importanza, facendolo operare come un essere "quasi intelligente", grazie alla sua capacità decisionale.

Un grande direttore: la CPU

Cominciamo ad analizzare le varie unità che compongono il sistema di un computer, partendo dal "cervello" del nostro sistema: la CPU. CPU sta per Central Processing Unit, che tradotto in italiano vuol dire Unità di elaborazione centrale. Essa in sostanza si prende cura, attraverso comandi adeguati, di far eseguire tutte le istruzioni necessarie al funzionamento del calcolatore e del programma.
Possiamo vedere quanto sia importante questa funzione facendo una analogia con un concerto sinfonico. L'insieme dell'orchestra è rappresentato da tutto il complesso dei circuiti del calcolatore ed il direttore d'orchestra dalla CPU. Il direttore (la CPU) legge lo spartito (il programma) e impartisce ordini all'orchestra, facendo in modo che gli strumenti (le varie parti del calcolatore) partano solo quando è loro richiesto e si arrestino se necessario. Il direttore, poi, cura che il tutto avvenga armonicamente, cioè che il programma si svolga secondo schemi precisi.
La CPU, insomma, "dirige" il computer.
Nonostante gli onerosi impegni che ricadono sulle sue spalle, la CPU, come aspetto fisico, non si discosta di molto da altri componenti posti all'interno di un calcolatore, tanto da passare quasi inosservata.
Sul mercato esistono diversi tipi di CPU, per soddisfare il più possibile le esigenze di chi le usa: alcune sono molto veloci, altre più potenti, altre ancora più flessibili, ma tutte simili a ragni neri dalle molte zampe, per altro difficilmente distinguibili a colpo d'occhio l'una dall'altra.
Per fortuna recano una sigla: Z80A le CPU degli MSX, 6510 quella del C64 e 7501 quella dei COMMODORE 16.
Le CPU tra loro (come le interpretazioni dei direttori d'orchestra) non sono equivalenti, ma richiedono istruzioni e circuiti diversi.
Una volta scelta la CPU sulla quale operare, le si progetta attorno il calcolatore e, grosso modo, vengono determinate le prestazioni del computer. Tutte le CPU, comunque devono risolvere un problema comune: come eseguire un insieme di comandi.
Chiariamo meglio il problema. Tutte le istruzioni hanno qualcosa in comune: somma, sottrai, accendi, spegni, possono essere considerate come casi diversi di un compito comune. Come dire che questi comandi sono considerati uguali ed analizzati allo stesso modo. La differenza sta nell'esecuzione; infatti la comprensione del comando è qualcosa di ben diverso dalla sua attuazione. In particolare una CPU considera allo stesso modo tutti i comandi ('traduce' il programma nelle sue istruzioni elementari) e li fa corrispondere ad azioni diverse. È come tradurre in inglese dall'italiano: esistono regole generali e ben definite di grammatica e

di sintassi, e si può fare una traduzione senza tener conto del significato reale dei singoli vocaboli. La traduzione di questi ultimi viene poi fatta a parte (l'esecuzione del comando, appunto).
In sintesi, ogni macchina che "fa conti" (e quindi una CPU) deve essere in grado di:
1) rappresentare, registrare e manipolare numeri;
2) identificare un'istruzione;
3) eseguirla;
4) scegliere quella successiva.
Per fare ciò sono state demandate ad alcune parti della CPU (l'ALU, il clock, i registri, la memoria di sola lettura e l'unità di controllo) delle funzioni ben precise.
Ma prima di vedere in dettaglio queste parti è meglio vedere come una CPU opera realmente.
Quando tu accendi il computer "dai vita" alla CPU, che inizia leggendo un programma contenuto al suo interno. Quest'ultimo ordina alla CPU di eseguire il programma che parte da un certo indirizzo e via via, una dopo l'altra, tutte le istruzioni passo-passo.
La CPU legge l'istruzione all'indirizzo

corrispondente, la confronta con una lista, contenuta al suo interno, di quelle che può eseguire e trova la "ricetta" corrispondente a quel comando. In ultima analisi ogni istruzione viene eseguita tramite una serie di operazioni elementari svolte dai circuiti preposti (interni alla CPU), i quali, come nell'esempio dell'orchestra, vengono attivati e disattivati in base al programma.
Per fare questo c'è bisogno della presenza delle unità citate prima, che sono:
1) l'ALU o Unità Aritmetico Logica, la quale svolge tutte le operazioni matematiche e logiche (come il confronto tra due numeri);
2) la ROM o memoria di sola lettura (interna alla CPU), dove sono contenute permanentemente le informazioni dei comandi;
3) i registri, che sono delle memorie temporanee dove la CPU può tenere dei numeri senza perderli;
4) il clock (l'orologio), che dà il tempo a tutte le parti del microprocessore, esattamente come il metronomo scandisce il ritmo al suonatore.
La ROM interna e il clock (visti congiuntamente) vengono comunemente indicati come unità di governo, a sottolineare la loro importanza nella gestione del "microsistema".

Il BUS

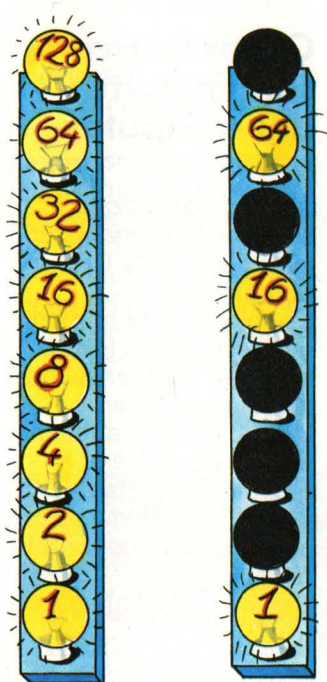
Un direttore d'orchestra, per comunicare con gli strumentisti, usa la bacchetta: ma quale 'bacchetta' usa il computer?

In altre parole, noi abbiamo detto fino ad ora cosa, e come, è in grado di fare la CPU, ma non come quest'ultima mette a disposizione degli altri circuiti i dati che essa elabora.
In ogni CPU esistono strutture preposte a comunicare i dati e a far conoscere agli altri circuiti le direttive della CPU.
Questi circuiti vengono chiamati, in gergo, bus, traducibile liberamente in italiano come "mezzo di comunicazione" tra CPU e il resto del computer. In ogni CPU ve n'è più d'uno, per poter far fronte a tutte le esigenze. Esigenze che

possono essere:
a) la necessità di comunicare quale cella di memoria la CPU sta analizzando (il cosiddetto bus degli indirizzi);
b) la necessità di far conoscere qual è il dato contenuto in quella cella (il bus dei dati);
c) la necessità di conoscere dove vengono immesse le informazioni relative al comando dei circuiti ausiliari (il bus di controllo).
I bus tuttavia non sono da intendersi come una vera e propria parte circuitale della CPU, ma piuttosto come una sorta di canale dove vengono inviate tutte le informazioni relative a un certo problema. È il bus, e non la CPU, che mette a disposizione questi dati a tutti i circuiti: la CPU si limita a metterli nel canale.

Bit

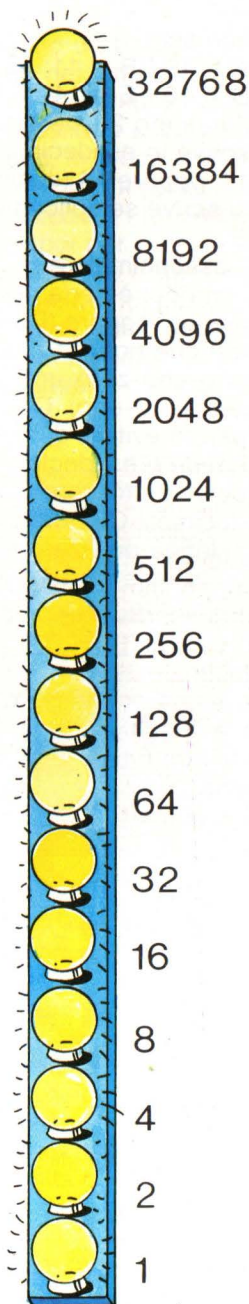
Già adesso intravediamo la "logica" della CPU: essa in sostanza, pur conoscendo solo due stati (accesso o spento), attiva o disattiva un certo numero di circuiti.
La CPU quindi adotta una logica binaria (a due stati).
Per brevità essi si indicano con 1 o 0.
Ognuna di queste cifre si chiama bit (da Binary digiT) e su di esse si fonda l'algebra binaria o di Boole, dal nome del matematico che la inventò.

Binario

Vediamo ora come si rappresentano i numeri in logica binaria.
Supponiamo di voler conoscere quanto vale in binario il numero decimale 39.
Vi è una premessa da fare. Il nostro sistema di scrivere dei numeri è a base 10, cioè utilizza dieci differenti simboli (0 ÷ 9), ed è un sistema posizionale: la stessa cifra, a seconda della posizione, assume un valore diverso (10^0, 10^1, 10^2... 10^n).
Quando scriviamo 39 diamo cioè per sottinteso la seguente relazione:
$39 = 3 \times 10^1 + 9 \times 10^0 =$
$= 3 \times 10 + 9 \times 1$
Ricordiamo che qualsiasi numero elevato a 0 è uguale a 1.
In binario abbiamo visto che gli stati possibili sono 1 e 0, quindi un sistema a base due (utilizza due simboli).
Potremo però scrivere come per i numeri decimali:
$00100111 = 0 \times 2^7 +$
$+ 0 \times 2^6 + 1 \times 2^5 +$
$+ 0 \times 2^4 + 0 \times 2^3 +$
$+ 1 \times 2^2 + 1 \times 2^1 +$
$+ 1 \times 2^0$.
Torniamo, dopo questa brevissima nota, al nostro problema di trasformare in binario il numero 39.
Si prende il numero da trasformare in binario, si guarda qual è la massima potenza di due contenuta nello stesso, la si sottrae dal numero precedente fino a che non si annulla. Nel nostro caso il procedimento è:
la massima potenza di due contenuta in 39 è 5:
$2^5 = 32$ $39-32 = 7$;
la massima potenza contenuta in 7 è 2:
$2^2 = 4$ $7-4 = 3$;
la massima potenza contenuta in 3 è 1:
$2^1 = 2$ $3-2 = 1$;
la massima potenza contenuta in 1 è 0:
$2^0 = 1$ $1-1 = 0$ FINE.
Dove non esiste la potenza di due va inserito 0 - otteniamo quindi: 100111.
Un altro metodo è quello di dividere successivamente per

due il numero decimale da convertire in binario, annotando, a partire dalla destra, i resti:

39 : 2 = 19 resto 1
19 : 2 = 9 resto 1
9 : 2 = 4 resto 1
4 : 2 = 2 resto 0
2 : 2 = 1 resto 0
1 : 2 = 0 resto 1

quindi 39 in binario corrisponde a 100111.
In binario in realtà abbiamo:

$39 = 0 \times 2^7 + 0 \times 2^6 +$

$+ 1 \times 2^5 + 0 \times 2^4 +$
$+ 0 \times 2^3 + 1 \times 2^2 +$
$+ 1 \times 2^1 + 1 \times 2^0 =$
$= 00100111.$

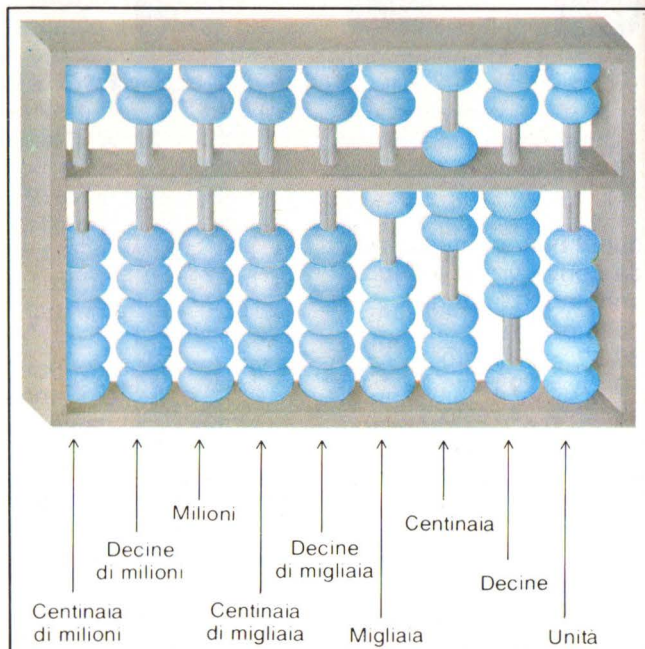
Il numero 39 in binario è stato rappresentato con 8 cifre (00100111) anziché 6.
È un po' come scrivere 039, il che non sembrerebbe avere molto senso. In realtà è molto importante per un computer; infatti questo considera i numeri di lunghezza standard, per ovvie esigenze pratiche. Nel caso del tuo MSX la lunghezza standard è di 8 bit, negli anni passati invece, quando la tecnologia non era al livello attuale, era di 4 bit. Già oggi per alcuni è di 16 e per altri di 32 bit. Siccome il linguaggio del computer si basa sui numeri, si è pensato di chiamare questo **raggruppamento: parola o byte.**
Il numero più alto rappresentabile con un byte è 11111111 = 255.
Ma se volessimo rappresentare un numero più alto?
Questo tipo di notazione è chiaramente insufficiente.
Basta però considerare due byte consecutivi in memoria come un unico numero per poter arrivare a 65535.

Esadecimale

Per non cambiare la struttura della CPU, per la rappresentazione di un numero si usano due byte in fila, considerando ogni informazione come composta da 16 bit. Tuttavia, poiché risulta poco pratico scrivere 16 bit tutti in fila per rappresentare un numero, è stato introdotto il sistema di numerazione in base 16 (esadecimale), analogo al nostro decimale con l'unica differenza che oltre ai dieci simboli usuali se ne utilizzano altri sei:
A = 10 B = 11 C = 12
D = 13 E = 14 F = 15.
Il numero 20, perciò, si scrive in esadecimale 14, oppure 14 (decimale) si scrive semplicemente E.
L'esadecimale è comodo: è certamente meglio leggere (D3 di-tre) che non (11010011 uno-uno-zero-uno-zero-zero-uno-uno). Occorre perciò evitare di leggere i numeri esadecimali come se fossero decimali. Quindi 20 (hex) si legge due-zero e non venti; vale infatti trentadue.

Cenni storici sulla nascita dei computer

Abbiamo analizzato in sintesi il funzionamento ed il ruolo della CPU in un calcolatore.
L'abbiamo fatto in modo da giustificarne le sue singolari caratteristiche (l'aritmetica binaria, i bus,...), ma non abbiamo sviluppato come si è arrivati ad una tale struttura. Tantomeno poi abbiamo chiarito il ruolo della logica.
Storicamente, infatti, la

L'abaco può essere considerato la prima macchina da calcolo.

logica matematica ha codificato i principi del calcolo matematico molto prima che il progresso scientifico e tecnologico rendessero possibile la realizzazione dei calcolatori. L'"idea" di computer era già nota molto prima che questi, nella comune accezione del termine, venissero costruiti.

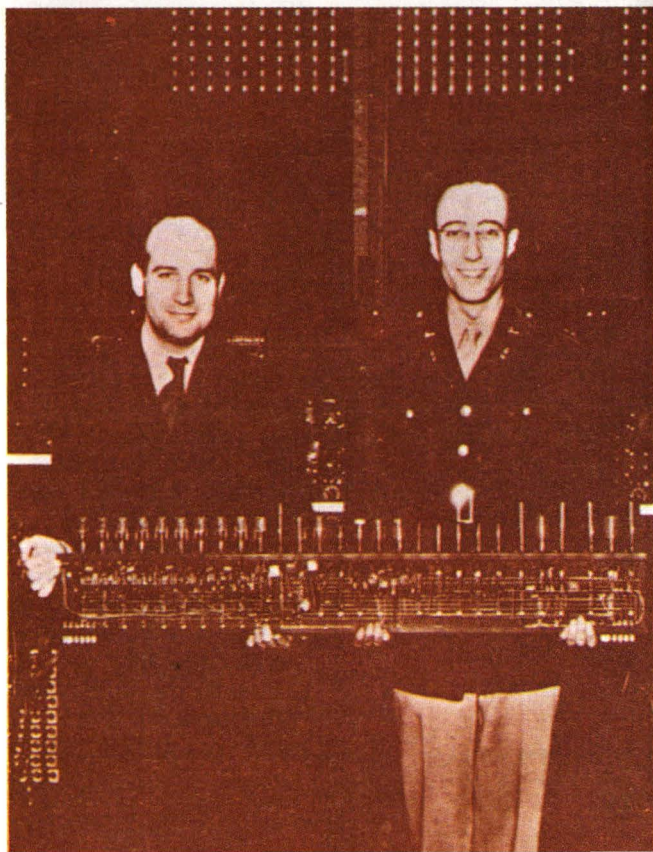
Facciamo un balzo indietro nel tempo. L'uomo ha sempre avuto la necessità di eseguire dei conti, in modo esatto e rapido: inizialmente per prevedere i fenomeni celesti, poi per determinare le rotte marittime, quindi per eseguire le complesse operazioni legate all'incremento dei traffici commerciali.

La Pascalina rese per la prima volta automatica l'operazione di riporto.

Si pensa che la prima macchina da calcolo inventata dall'uomo sia stato l'abaco, il comune pallottoliere ancora oggi impiegato in alcune nazioni. L'idea di base era quella di espandere, al di là del numero 10, le possibilità di conteggio offerte dalle dita.
Si dovette però attendere i primi anni del XVII secolo per arrivare alla prima vera macchina calcolatrice meccanica, opera del matematico e pensatore francese Pascal. Benché la pascalina (questo il nome datole) fosse in grado di eseguire solo somme e sottrazioni, è stata a tutti gli effetti il primo esempio di calcolatore: data una informazione in entrata, mediante complicate rotazioni degli ingranaggi si otteneva un risultato in uscita. Certo, anche se siamo ben lontani dal concetto di calcolatore che abbiamo oggi, era la prima risposta fornita dall'uomo al bisogno di calcoli automatici. Mancava ancora il concetto di sequenza di comandi, e quindi di controllo.
Il decisivo passo in avanti fu opera del logico inglese Charles Babbage, che teorizzò nel 1834 la Macchina Analitica.
Nella macchina analitica, basata su un programma a nastro di carta perforata (ideato da Jacquard per la programmazione dei telai), possiamo già riconoscere le parti fondamentali di un computer moderno. Vi era l'unità di calcolo,

chiamata fabbrica o mulino, dove venivano eseguiti i calcoli, e la memoria centrale, detta magazzino.

Lady Ada Lavelace, amica di Babbage e sua collaboratrice, a questo proposito affermò: "Possiamo dire che la Macchina Analitica tesse disegni algebrici proprio come il telaio di Jacquard tesse fiori e foglie". Per la sua attività Lady Lavelace è considerata la prima programmatrice della storia.
Babbage però fu sconfitto dalla inadeguatezza della tecnologia di allora e riuscì solo in parte a costruire la sua Macchina Analitica.
Dal nastro perforato si passò con Hollerith (il fondatore della IBM) alle schede perforate.
Per avere il primo vero computer, però, bisogna aspettare il 1946, anno in cui nacque l'ENIAC: Calcolatore e Integratore Numerico Elettronico.
Per la prima volta, ma da

Goldstine e Eckert con una unità a tubi dell'ENIAC.

allora le loro storie avrebbero viaggiato assieme, l'elettronica, con le valvole, entrava in un computer.
Migliaia di valvole, che dovevano essere sostituite ogni 6 ore pena la bruciatura (si riscaldavano a tal punto da rendere necessari impianti di raffreddamento), non consentivano elevate velocità di calcolo (pensa che il tuo MSX è molto più potente, veloce e versatile dell'ENIAC, nonostante quest'ultimo occupasse la superficie di un'intera palestra).
Nel 1948 J. Bardeen, W. Brattain e W. Schockley annunciano la scoperta di un componente fondamentale: il transistor.
Il transistor compie tutte le funzioni di una valvola, ma con numerosi vantaggi: occupa meno spazio, dissipa poco calore, è molto più rapido, è quasi eterno.
Era inevitabile che venisse applicato ai computer, dando origine alla cosiddetta "nuova generazione".
Grazie al minor costo, poi, i calcolatori ebbero una grande diffusione ed incominciò allora la "battaglia" delle prestazioni, una incruenta lotta per il miglioramento delle caratteristiche.
Dal transistor in poi l'elettronica concentrò tutti i suoi sforzi sull'integrazione: miniaturizzazione e compattazione dei componenti base (tra i quali il citato transistor) in uno spazio sempre più ridotto.
Ridurre i componenti a dimensioni visibili solo al microscopio non è affare da poco ed il cammino dal 1958 (anno in cui fu costruito il primo circuito integrato) ad oggi è costellato di successivi miglioramenti:

- anni '60, SSI (piccola scala di integrazione) 12 porte logiche per circuito integrato;
- fine anni '60, MSI (media scala di integrazione) 100 porte logiche per circuito integrato;
- anni '70, LSI (grande scala di integrazione) 1000 porte logiche per circuito integrato;
- fine anni '70, VLSI (grandissima scala di integrazione) oltre le 50.000 porte logiche per circuito integrato.

Oggi in 1 cm^2 di superficie possono risiedere oltre 100.000 transistor.
La storia del computer e dell'elettronica, comunque, è ben lontana dall'esaurirsi; si è calcolato infatti che il progresso nel campo dell'integrazione non ha ancora raggiunto il suo massimo livello, e c'è da aspettarsi una nuova "rivoluzione" nel campo.
Il tuo MSX, tuttavia, è figlio di questa tecnologia e senza quest'ultima non sarebbe in grado, in così poco spazio ed a basso prezzo, di compiere tutte le funzioni che via via imparerai a conoscere.

BIN$

È un particolare tipo di istruzione chiamato funzione, in grado di fornirti una rappresentazione binaria (si tratta in realtà di una stringa) del numero racchiuso tra due parentesi tonde.

```
PRINT BIN$ (13)
```

Per ottenere, invece, il numero decimale corrispondente ad un certo numero binario, puoi chiedere al tuo MSX la stampa del numero binario preceduto dal prefisso &B.

```
PRINT &B1101
```

Ricorda di non lasciare spazi tra il prefisso ed il numero!

HEX$

È la funzione che ti permette di ottenere automaticamente la rappresentazione esadecimale di un dato numero sotto forma sia di costante che di variabile.

```
PRINT HEX$ (13)
```

oppure

```
LET A = 13 : PRINT HEX$ (A)
```

oppure

```
PRINT HEX$ (&B1101)
```

Per convertire in decimale un numero esadecimale, usa il prefisso &H

```
PRINT &HD
```

OCT$

Questa funzione restituisce la rappresentazione ottale del numero fornito tra parentesi.

```
PRINT OCT$ (9)
```

Anche in questo caso puoi richiederle la conversione in decimale di una costante ottale (con prefisso &0).

```
PRINT &O11
```

REM

La REM (abbreviazione del termine inglese Remark, commento) è una istruzione ... che sembra non serva a niente!
Quando il tuo MSX incontra una REM, infatti, la ignora completamente e passa ad eseguire la linea successiva del programma.
In realtà, REM è una istruzione importante: ti permette di inserire commenti, titoli e descrizioni, senza che

questi alterino il normale svolgimento del programma.
Il suo argomento può essere una sequenza qualsiasi di caratteri, anche simboli grafici e matematici, anche una riga vuota.
Puoi inserire una REM in qualsiasi punto del programma, o in fondo a ogni linea di istruzione, basta che rispetti le regole di sintassi del tuo MSX.
Una cosa ancora. Dopo la REM non puoi più inserire alcuna istruzione o comando, in quanto verrebbe considerato commento.

Esempi

```
10 REM "GEOMETRIA 1"
```

Questa linea dà il titolo "GEOMETRIA 1" al tuo programma. Quando, a distanza di tempo, lo riprenderai in mano non dovrai analizzare le istruzioni per capire se il programma in questione è una contabilità domestica o un archivio dei dischi. Hai la risposta alla prima linea del programma.

```
60...
70 INPUT B
80 REM CALCOLO
   DELLA MEDIA
90...
```

Ti comunica che le linee successive alla REM servono al calcolo della media.

```
80...
90 LET M = (B + C + F + G)/4
100...
```

Quando ti occorrerà calcolare la media di più numeri per qualche altro programma saprai dove andare a recuperare le istruzioni necessarie senza riscriverle.
Oppure: se al posto della media devi ottenere la radice quadrata di quei numeri, potrai semplicemente sostituire le linee seguenti la REM, senza per questo dover rianalizzare e modificare tutto il programma.

```
30 REM "GEOMETRIA" : PRINT GEOMETRIA
```

Il tuo MSX interpreta tutto l'argomento di REM ("GEOMETRIA" : PRINT GEOMETRIA) come un commento e non eseguirà mai l'istruzione contenuta (PRINT).

Sintassi dell'istruzione

REM sequenza qualunque di caratteri.

GOTO

Parlando di algoritmo hai visto che esso contiene tutte le informazioni necessarie a portare a termine una elaborazione.
L'algoritmo, però, non deve per forza svilupparsi sequenzialmente (una istruzione dopo l'altra), ma può procedere anche a "salti", in modo da modificare all'occorrenza l'ordine delle istruzioni da eseguire. Il BASIC possiede una istruzione che svolge questo compito: GOTO.
L'istruzione GOTO (salta a ...) è sempre seguita da un numero in forma esplicita. In pratica, quando il tuo MSX incontra una GOTO, anziché proseguire con l'istruzione successiva salta a quella indicata, proseguendo da questa come se nulla fosse (attento, però, perché se quest'ultima non esiste, l'interprete BASIC ti invierà il messaggio di errore <Undefined line number>, per avvertirti appunto che quella linea non c'è).
Ti è insomma possibile, mediante GOTO, modificare l'ordine di esecuzione in base alle tue esigenze.
Un chiaro esempio è il calcolo della tabella dei quadrati:

```
10  REM CALCOLA I QUADRATI DI NUMERI
20  REM A PARTIRE DA 1
30  REM FINO A QUANDO NON LO SI FERMA
40  LET NUMERO =0
50  LET NUMERO = NUMERO +1
60  REM AGGIUNGE 1 AD OGNI PASSAGGIO
70  PRINT NUMERO, NUMERO ↑ 2
80  REM STAMPA IL NUMERO E IL SUO QUADRATO
90  REM RIPETE INCREMENTANDO
100 GOTO 50
110 REM PER INTERROMPERE PREMI I TASTI CTRL E STOP
    CONTEMPORANEAMENTE
```

LINGUAGGIO

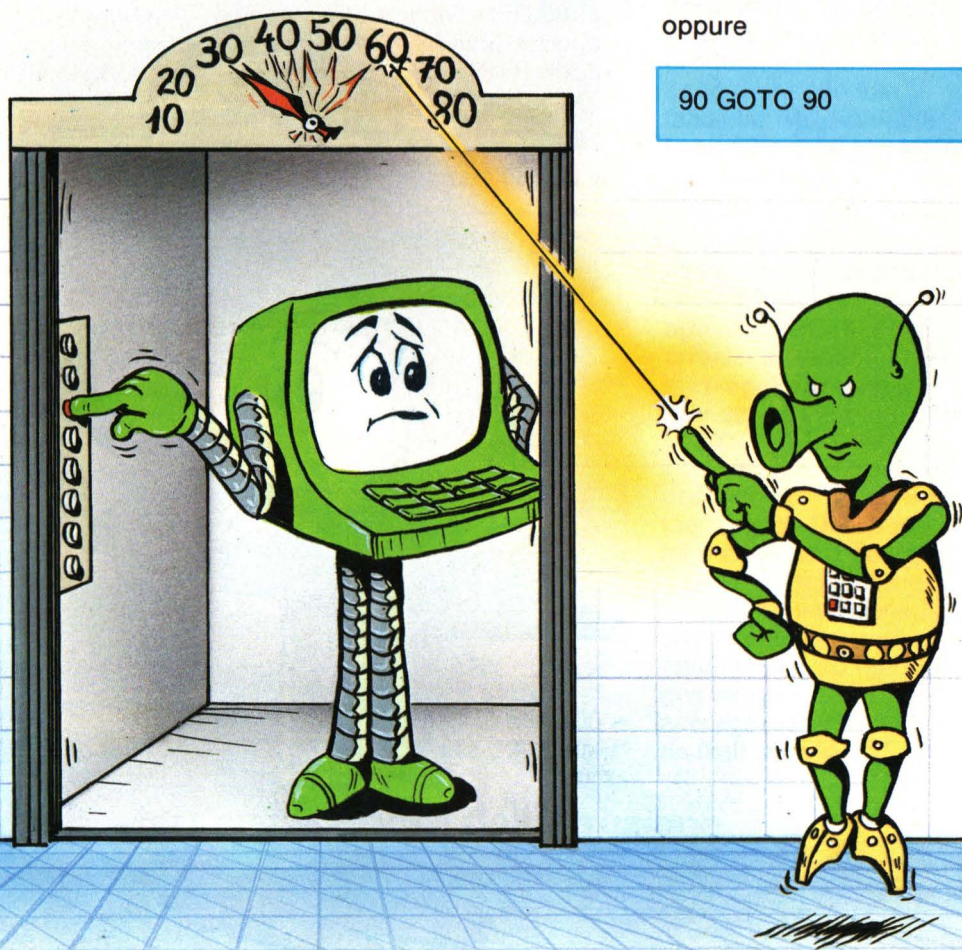
GOTO viene detta anche istruzione di salto incondizionato, poiché ordina al MSX di saltare alla riga desiderata in qualunque caso (incondizionatamente). Ad esempio, osserva il programma che segue:

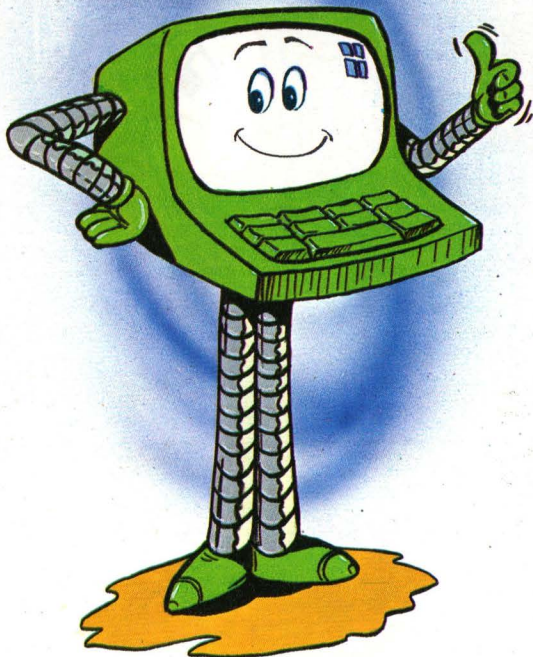
```
10 GOTO 40
20 .....
30 .....
40 REM
```

Il programma inizia sempre dalla riga 40, saltando le linee 20 e 30. Questa è la sua applicazione più comune. Ci sono comunque alcuni casi particolari. Uno di questi è:

```
90 istruzione qualsiasi: GOTO 90
```

oppure

```
90 GOTO 90
```

La prima viene usata in casi particolari per interrompere il flusso del programma senza che quest'ultimo si fermi realmente: il calcolatore esegue un numero infinito di salti alla riga 90.
La seconda è equivalente a 90 GOTO 90, con l'unica differenza che il tuo Computer esegue un numero infinito di volte i comandi specificati prima dell'istruzione GOTO.
Ambedue, comunque, servono per creare programmi che non si fermano mai, se non per un comando esterno.
Provare, per credere, questo programma:

```
10 REM PROGRAMMA CHE SI INTERROMPE
20 REM SENZA FERMARSI
30 PRINT "SONO ALLA RIGA 30"
40 PRINT "SONO ALLA RIGA 40" : GOTO 40
50 PRINT "QUI NON ARRIVA"
```

Esempi

```
80 GOTO 5000
```

Il tuo MSX dopo essere arrivato alla linea 80 passa direttamente alla 5000 trascurando tutte le linee intermedie.

```
10 REM È UN QUADRATO?
20 INPUT "NUMERO LATI UGUALI"; L
30 INPUT "NUMERO ANGOLI UGUALI"; A
40 GOTO 20
50 PRINT "È UN QUADRATO"
60 PRINT "NON È UN QUADRATO"
```

L'argomento dell'istruzione GOTO va scelto con cura: un suo valore errato può avere conseguenze catastrofiche sul tuo programma. Se consideri infatti questo semplice programmino, ti accorgerai che alle fatidiche istruzioni alle linee 50 e 60 il tuo MSX non arriverà mai.

```
10 PRINT "BIT"
20 PRINT "SUPERBIT"
30 GOTO 15
40 END
```

Il tuo MSX dopo aver stampato una sola volta BIT e SUPERBIT, arrivato a GOTO 15 si arresta, inviandoti il messaggio di errore: <Undefined line number in 30>.

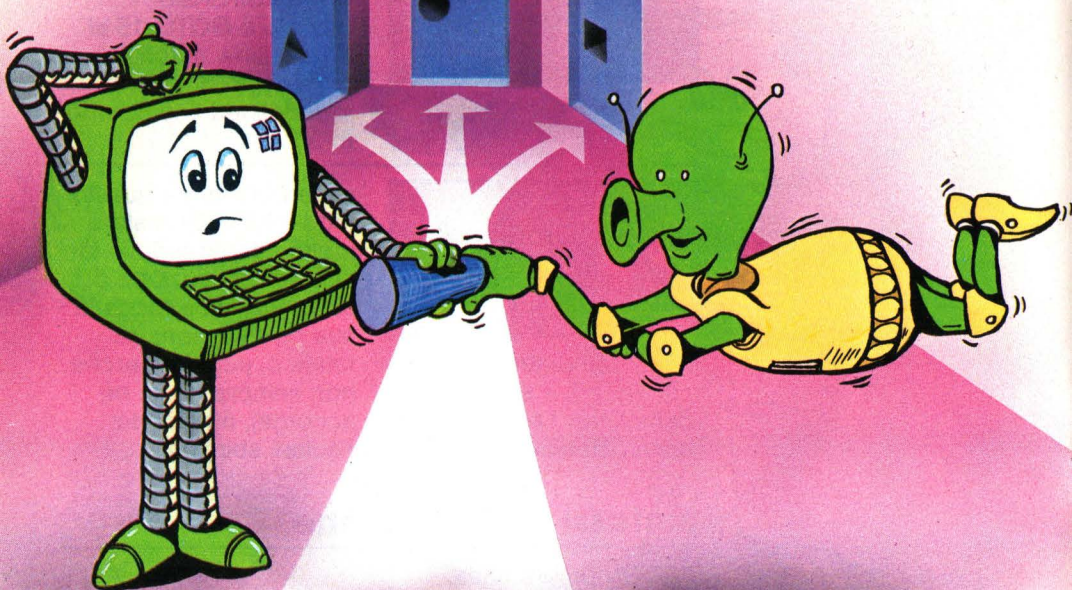
```
130 REM
140 GOTO 300
150...
300 REM
310 GOTO 130
```

Il tuo computer continuerà a saltare dalla linea 140 alla 300 e dalla 310 alla 130, non uscendo mai da questo loop (ciclo). Devi stare molto attento a dove far saltare l'esecuzione di un programma, pena la sua non funzionalità totale o, peggio (perchè non sempre ce se ne accorge), ottenendo risultati sbagliati.

Sintassi dell'istruzione

GOTO numero linea.

IF THEN ELSE

Come avremo occasione di puntualizzare più avanti (quando si continuerà il discorso sui flow-chart), una parte basilare della programmazione si occupa delle decisioni. È grazie alle decisioni che riusciamo ad adattare una soluzione generale (il programma) alla situazione particolare (il problema da risolvere). Senza qualcosa che permetta al programma di cambiare il suo corso in base al contesto, molte applicazioni del

computer non sarebbero possibili: bisognerebbe scrivere un programma rigidamente sequenziale. Nella nostra vita, del resto, le decisioni ricoprono un ruolo fondamentale: soltanto che noi ne sottovalutiamo l'importanza, perchè abituati da sempre a prenderne.
Il computer, è importante sottolinearlo ancora, è un formidabile esecutore a cui bisogna dire tutto quello che deve fare (anche le cose che sembrano più ovvie), dato che manca di raziocinio e di iniziativa propria.
La decisione quindi è uno dei cardini della programmazione.
Cosa può decidere un computer? Parlando della CPU abbiamo detto che il computer analizza solo e soltanto numeri e sa dire, sempre tra due numeri, se questi sono uguali oppure diversi.
Per ridurre la decisione in maniera "capibile" dal computer bisogna quindi ridurre il problema in numeri.
Se numero 1 = numero 2, allora scrivi "i due numeri sono uguali", altrimenti...
Questa è la forma computabile, e in BASIC si scrive

```
IF CONDIZIONE LOGICA THEN ISTRUZIONE
ELSE ALTRA ISTRUZIONE
```

Supponi di aver impostato la variabile X con un numero segreto che intendi far indovinare da un amico e di inserire nella variabile T il tentativo di risoluzione. Ciò di cui hai bisogno è di una istruzione che ordini al tuo MSX di segnalare quando il numero segreto è stato indovinato, ossia quando la variabile T è uguale alla variabile X.
A questo punto scriverai:

```
SE T = X ALLORA STAMPA "OK"
ALTRIMENTI STAMPA "RIPROVA"
```

Bene. Non devi fare molta fatica ad immettere

una istruzione del genere, lo hai già fatto ... È sufficiente che tu traduca in inglese l'istruzione, così come l'hai pensata, e avrai:

```
IF T = X THEN PRINT
"OK" ELSE PRINT
"RIPROVA"
```

Risulta evidente l'importanza di IF ... THEN ... : serve a far compiere al computer delle vere e proprie scelte, in base alle quali il programma può eseguire, di volta in volta, la funzione più adeguata alla particolare circostanza. È un po' come dare al tuo MSX capacità di discernimento, quasi un "cervello".
Naturalmente, anziché PRINT puoi inserire qualsiasi altro comando, come, del resto, puoi immettere comparazioni per maggiore o minore su valori numerici od alfanumerici.
Qualora la condizione non sia vera, viene ignorato completamente il seguito della linea fino all'eventuale parola chiave < ELSE > ed il controllo passa alle istruzioni poste di seguito a quest'ultima o, in mancanza, alla linea di programma successiva.
Ricorda, quindi: tutta la parte susseguente il THEN fino ad un eventuale ELSE è soggetta alla comparazione.
Questa particolare possibilità offre dei vantaggi notevolissimi: ti permette infatti di scrivere gruppi completi di istruzioni controllati da un solo IF, risparmiando superflui GOTO ed ottenendo programmi compatti, veloci e molto leggibili.
Ti capiterà comunque molto spesso di trovare IF ... THEN GOTO, in quanto la combinazione di queste due istruzioni genera vere e proprie diramazioni di programma, molto simili, come concetto, a scambi ferroviari, i quali, a seconda dell'impostazione, dirigono verso l'uno o l'altro percorso.
Vediamo una semplice ma utile applicazione pratica di IF THEN GOTO, riprendendo uno degli esercizi visti (90 GOTO 90).
Apportandogli una piccola modifica:

```
80 PAUSA = 0
90 PAUSA = PAUSA + 1 = IF PAUSA < 1000 THEN GOTO 90
```

Otteniamo un "timer" : il tuo computer infatti conta fino a 1000 (in questo caso) prima di proseguire con l'istruzione successiva.
Questo può essere un valido espediente quando hai la necessità di ritardare l'esecuzione di alcune istruzioni.

Nel caso poi debba paragonare valori alfanumerici, il calcolatore assegnerà ad ogni lettera dell'alfabeto, simbolo grafico, ecc. un determinato numero, secondo un preciso standard. Questo codice è proporzionale alla posizione alfabetica del carattere: ad esempio, A corrisponde a 65, B a 66, C a 67 ... e così via. A questo punto il confronto diviene semplicissimo, riducendosi alla comparazione dei codici che compongono le due stringhe.
Dopo un'istruzione IF THEN puoi omettere la parola riservata GOTO, indicando solo il numero di linea a cui saltare. Il risultato sarà lo stesso un salto alla linea indicata.

Esempi

```
10 PRINT "QUANTI SOLDI HAI IN TASCA?"
20 INPUT SOLDI
30 IF SOLDI ≥ 30000 THEN PRINT "SEI RICCO!"
40 IF SOLDI < 30000 THEN PRINT "NON NE
   HAI MOLTI!"
50 IF SOLDI = 0 THEN PRINT "NON HAI NIENTE!"
60 END
```

Analizziamo il programma: se una delle tre espressioni è vera, il programma stampa il commento appropriato. Se avessi 5000 lire, il risultato sarebbe: "NON NE HAI MOLTI!" poiché la seconda istruzione IF è vera.

```
70 IF TV = 30 THEN LET RADIO = 24
```

Se la variabile TV vale 30, allora la variabile radio assume il valore 24.

```
70 IF NUMERO = 88 THEN IF ALTRNUMERO
   = 66 THEN GOTO 90
```

Il comando IF THEN ha alcune caratteristiche interessanti, come la possibilità di concatenare più di una istruzione in sequenza. Questo processo si chiama nidificazione, poiché raggruppa (annida) più istruzioni come parte di una istruzione sola. Nel caso specifico se il numero è 88, il calcolatore controlla che l'altro sia 66.

Virtualmente non c'è limite alla nidificazione di IF THEN; tuttavia è bene non abusarne per evitare problemi di correzione (DEBUGGING).

```
IF RUOTE = 4 AND VEICOLO = 1 THEN
PRINT "È UNA AUTOMOBILE"
```

IF controlla che una espressione logica sia vera: puoi allora inserire degli operatori logici che ti consentono di ampliare il controllo della IF a più di una espressione, come si è fatto in questo esempio. L'espressione RUOTE = 4 AND VEICOLO = 1 è vera solo quando ambedue le eguaglianze sono verificate, e solo in questo caso il calcolatore stampa il commento appropriato.

```
40 IF SOLDI < 1000 THEN PRINT "NON NE
   HAI MOLTI"
```

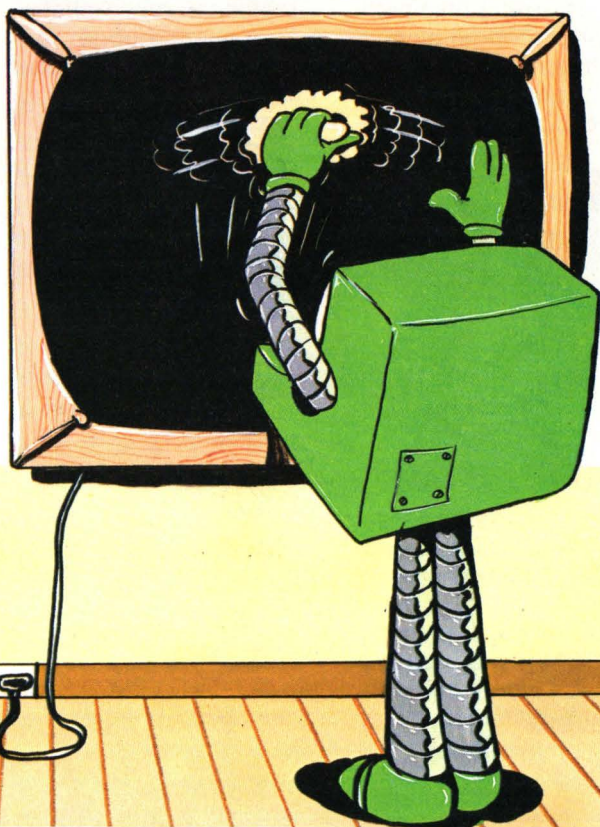
Devi fare attenzione alla scelta dell'argomento di IF. Se nell'esempio "QUANTI SOLDI HAI IN TASCA?", metti questa linea alla riga 40 e dici che ne hai 25000, il calcolatore, come puoi controllare, non ti risponderà niente!

Sintassi dell'istruzione

IF espressione logica THEN istruzione eseguibile.

CLS

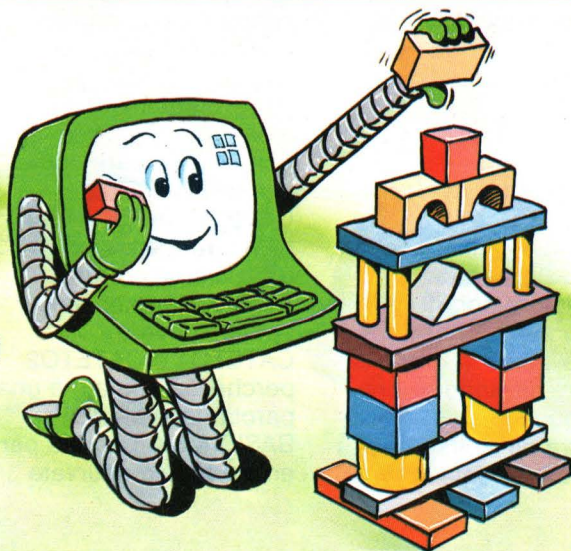
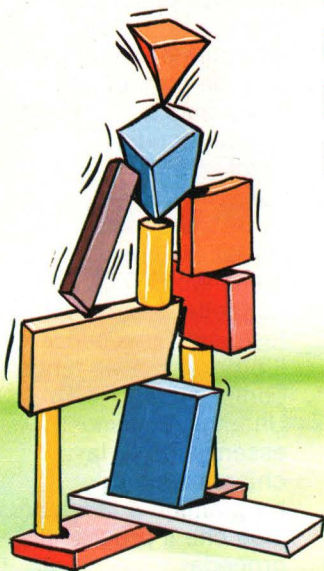
Sovente ci capita di dover scrivere ciò che diciamo, vuoi per annotare qualcosa vuoi quando occorre mostrare ad altri quanto fatto.
Altrettanto spesso siamo nella condizione di dover cancellare quello che abbiamo scritto perché non serve più o perché sbagliato.
Su un foglio bisogna tirare una croce o cestinarlo, su una lavagna passare con il cancellino.
In un computer, che considera lo schermo del TV come "foglio elettronico", basta dare il comando giusto.
Nel nostro caso CLS dall'inglese Clear Screen, pulisci lo schermo. Con queste sole tre lettere potrai pulire a piacimento il tuo TV di quanto già presente. Il tuo MSX considera lo schermo come un insieme di tanti piccoli punti; non fai fatica a vederli avvicinandoti allo schermo su cui hai scritto qualche cosa.
Quando incontra l'istruzione CLS "pittura" questi punti nello stesso colore del fondo (in pratica li cancella) e riporta il cursore nell'angolo in alto a sinistra.
Se in un programma devi visualizzare molte cose, non fare economia di CLS: i tuoi occhi ti ringrazieranno.

Effetto analogo al comando CLS si ottiene in modo immediato premendo contemporaneamente i tasti SHIFT e HOME oppure CTRL e L.

Come scrivere i programmi: tecniche di correzione e di chiarezza

Abbiamo già visto alcuni semplici programmi, ma ... quali sono i requisiti necessari per scrivere un buon programma? A un programma "ideale" di solito chiediamo di compiere tutte le operazioni necessarie nel minor tempo possibile, di essere facilmente modificabile e correggibile e di occupare la minor quantità possibile di memoria. Non siamo però in grado, se non in casi particolari, di poter variare di molto la velocità di calcolo, e di fatto il solo parametro su cui possiamo influire significativamente rimane la correggibilità del listato. Il compito più ingrato, infatti, nella stesura di un programma è di solito quello di trovare e correggere gli errori commessi, operazione chiamata, in gergo, debugging (tradotto letteralmente "spulciamento").

Il problema è ancora più sentito laddove il linguaggio offre molte (troppe) possibilità di approccio al problema stesso, come è appunto il BASIC. Spesso, non sapendo in quale zona del programma è celato l'errore, si deve analizzare il listato dall'inizio alla fine, con grande dispendio di tempo e fatica, proprio perché in esso sono contenuti, disordinatamente, variabili e istruzioni.
Il primo passo per una migliore stesura dei programmi è dunque dare un ordine al suo interno: creare cioè una struttura.
È buona norma, dunque, inserire per prima cosa tutte le variabili, in modo che siano immediatamente disponibili per un eventuale controllo e/o correzione. Così facendo, tra l'altro, aumenteremo leggermente la velocità di esecuzione del programma, dato che nel suo svolgimento il calcolatore non dovrà che leggere istruzioni "attive", cioè in grado di portare all'elaborazione dei dati in esame.
Un'altra difficoltà che si incontra durante la redazione di un programma è la sua astrattezza. Quasi tutti i linguaggi esistenti, infatti, sono legati, in misura più o meno accentuata, al modo in cui la macchina elabora le informazioni che noi le diamo. E questa "logica" è fondamentalmente molto lontana dal nostro modo di agire e di pensare, obbligandoci a "tradurre" la nostra logica in quella del calcolatore, lavoro non del tutto agevole. È consigliabile quindi adottare vari espedienti per rendere meno astratto e formale un programma, e quindi più vicino a chi lo utilizza.
Una prima tecnica consiste nell'utilizzare, nella dichiarazione di una variabile, dei nomi composti, che richiamino alla mente quale sarà il loro utilizzo. Ad esempio, per programmare il teorema di Pitagora potresti utilizzare come variabili CATET1, CATET2, IPOTENUSA; a, b, c sarebbero stati ugualmente corretti.
Attento. Non abbiamo potuto scrivere CATETO1 e CATETO2 perchè il TO finale è una parola riservata del BASIC MSX. Usando per errore parole riservate otteresti come risulato il messaggio < ? Sintax error in ... >e di bloccare l'esecuzione del programma.
Anche se non tutti i calcolatori prevedono questo tipo di variabili, i tuo MSX è fortunatamente in grado di riconoscere variabili composte.
Un altro tipo di strategia da adottare è quello di inserire nei punti-chiave del programma commenti ed indicazioni su quello che il blocco di istruzioni interessate è destinato a fare. In questo modo anche a distanza di tempo ti basterà un colpo d'occhio per riconoscere le varie parti di un programma, guadagnando in chiarezza e modificabilità.
In BASIC questa possibilità ti è data dal comando REM, che permette di scrivere un testo qualsiasi senza che il programma ne tenga conto.
Un altro requisito essenziale per la chiarezza di un programma è la linearità, cioè che il programma proceda sequenzialmente, una riga dopo l'altra, con il minor numero possibile

di "salti". Nel linguaggio BASIC l'istruzione "salto ad una riga" è GOTO e dovrai cercare di farne il minimo uso possibile, in modo da facilitare al massimo il debugging. Se il programma da te ideato e scritto prevede diverse possibilità di calcolo, in base alle necessità di chi lo usa, è bene dividere subito le varie possibilità, separandole in moduli distinti all'interno del listato.
Sempre per esigenze di chiarezza è utile inoltre scrivere linee di programma non troppo lunghe, per evitare di complicare la lettura.
Riassumendo i concetti sopra esposti, possiamo ora scrivere un programma BASIC che si avvicini di più alle nostre esigenze.
Riprendiamo, come esempio, il programma della superficie di un triangolo.
Iniziamo ponendo come prima linea del listato una REM che ci espliciti la finalità del programma.
Proseguiamo, poi, inserendo altre REM, che ci ricorderanno cosa fanno i vari moduli del programma. Otteniamo a questo punto:

```
10 REM LA SUPERFICIE DI UN TRIANGOLO
20 REM INTRODUZIONE DEGLI ELEMENTI
30 INPUT "BASE =  "; BASE
40 INPUT "ALTEZZA =   "; ALTEZZA
50 REM CALCOLO DELLA SUPERFICIE
60 LET AREA = BASE * ALTEZZA/2
70 REM STAMPA DEL RISULTATO
80 PRINT "L'AREA È  "; AREA
```

Come avrai notato, le variabili impiegate utilizzano nomi estesi, che dichiarano esplicitamente il loro contenuto.
Una legenda del programma potrebbe essere:

10 INTESTAZIONE
20 ÷ 40 MODULO INGRESSO DATI
50 ÷ 60 MODULO ELABORAZIONE
70 ÷ 80 MODULO USCITA DATI.

Le prime volte ti riuscirà artificioso programmare per strutture, ma dopo qualche tempo ne apprezzerai i vantaggi, quali la chiarezza dei listati, il risparmio di tempo e fatica durante e dopo la programmazione.
Il complesso di regole che consentono di cambiare un programma, aggiungendo e/o correggendo delle sue parti, si chiama EDITOR (in italiano qualcosa di simile a "correttore", "supervisore").
Ogni computer (anche il tuo MSX) ha un programma di editor che consente di compiere queste modificazioni: e delle sue possibilità potrai documentarti leggendo il Manuale di Istruzioni. È bene impadronirsi subito di questi strumenti di correzione, in modo da risparmiare fatiche inutili durante e dopo il debugging.

Le decisioni

A questo punto devi imparare a comunicare con il computer, magari per fargli eseguire qualche semplice operazione aritmetica.
Ma il suo valore e le sue capacità sarebbero effettivamente limitate se si riducessero esclusivamente a riprodurre modelli già precostituiti. Difatti poi possiamo parlare di "intelligenze artificiali", proprio perché i computer "decidono" cosa fare (riguardo alla realizzazione di compiti che noi affidiamo loro attraverso i programmi) in base a criteri logici e di valore, questi sì!, precedentemente precostituiti.
In BASIC, le decisioni sono eseguite usando l'istruzione IF...THEN, che permette al tuo MSX di fornirti il risultato di un TEST e quindi di eseguire una istruzione successiva invece di un'altra.

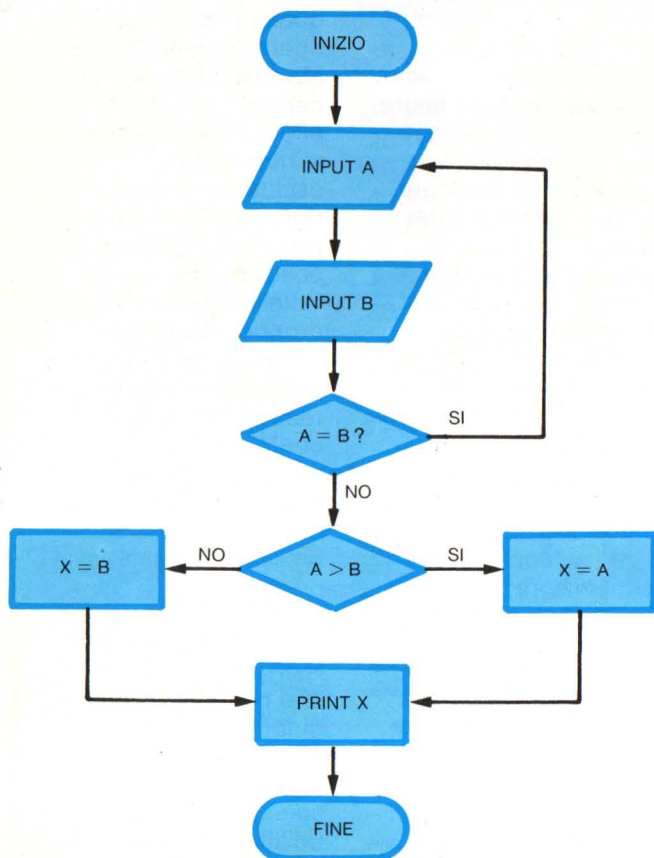
PROGRAMMAZIONE

Presentiamo ora un primo, semplice problema per vedere come puoi stendere un programma in cui il computer opera una "scelta" in base alle "conoscenze" che possiede in memoria.

Obiettivo

Vogliamo per esempio far determinare al nostro computer quale tra due numeri qualsiasi, che denominiamo A e B, sia quello maggiore. Vogliamo quindi che il computer faccia assumere ad una incognita, che chiamiamo X, il valore più alto.

Procedura

Per risolvere questo problema, bisogna dapprima raffrontare A e B. Se A > B, allora si pone il valore di A nella X, altrimenti nella X si pone il valore di B. Questo modo di procedere nella risoluzione del problema può essere rappresentato grazie al

Chiede il primo numero

Chiede il secondo numero

Se sono uguali, torna indietro, altrimenti ...

controlla quali dei due è il maggiore ...

... e lo stampa

flow-chart che contiene due "rombi", nei quali si trova il confronto tra A e B, e due "rettangoli", che corrispondono a delle "assegnazioni".
Il listato corrisponde alla sequenza BASIC relativa.

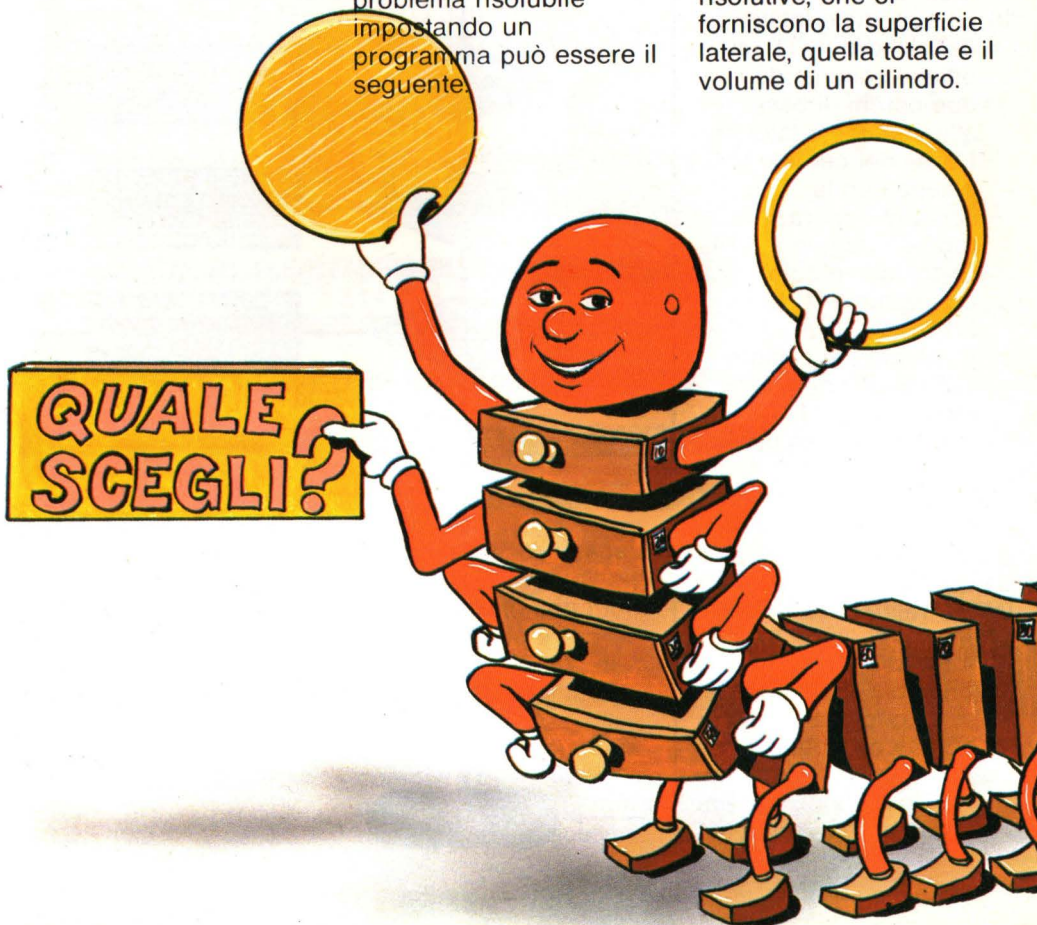
```
10 REM IL NUMERO MAGGIORE TRA 2
20 INPUT "PRIMO NUMERO = "; A
30 INPUT "SECONDO NUMERO = "; B
40 IF A = B THEN GOTO 20
50 IF A > B THEN LET X = A ELSE X = B
60 PRINT "IL NUMERO MAGGIORE È"; X
```

Un altro esempio di problema risolubile impostando un programma può essere il seguente:

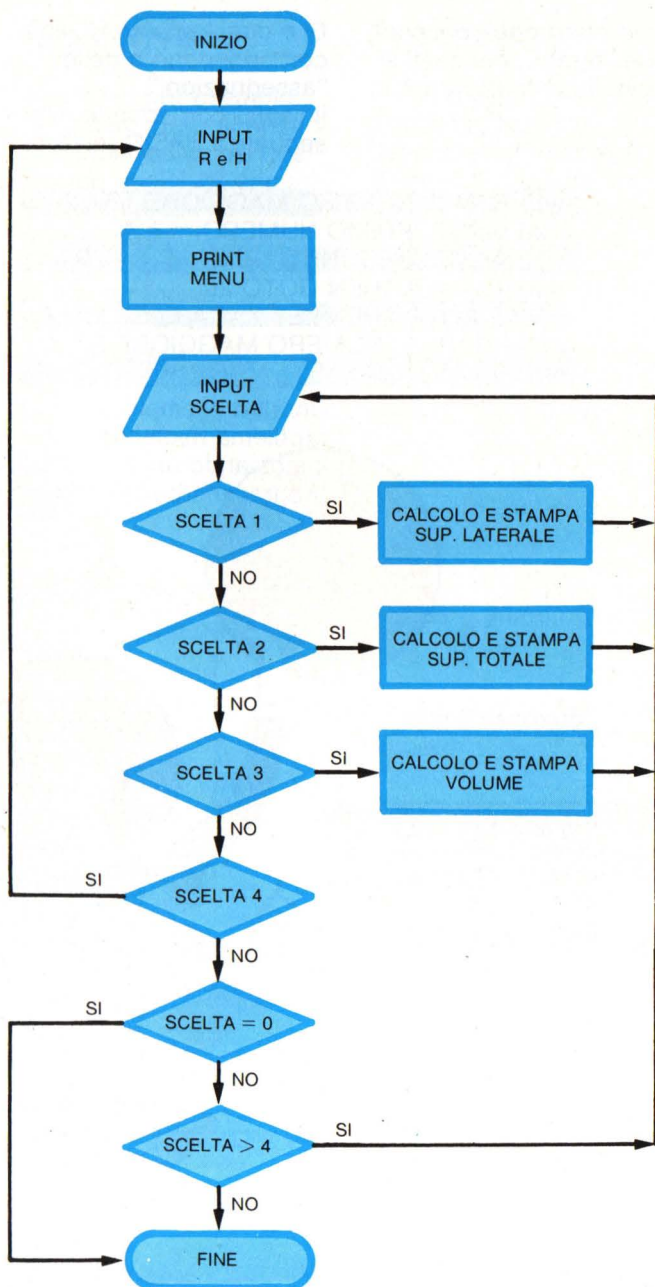
Obiettivo

Vogliamo determinare la superficie laterale, quella totale ed il volume di un cilindro di qualsiasi dimensione.

Procedura

Per poter affrontare questo problema bisogna conoscere le formule matematiche risolutive, che ci forniscono la superficie laterale, quella totale e il volume di un cilindro.

Vediamole assieme.

a) La superficie laterale del cilindro (SLAT) si ottiene moltiplicando l'altezza per la circonferenza di base, ricordando che quest'ultima è il risultato della seguente operazione: raggio x 2 x 3,14. Espressione questa che risulta, nel linguaggio del tuo MSX, R * 2 * 3.14.
b) La superficie totale del cilindro (SOT), si ottiene sommando i due cerchi di base alla superficie laterale. L'area del cerchio si ottiene con la seguente operazione: raggio x raggio x 3,14. Espressione cui corrisponde R↑ 2 * 3.14.
c) Il volume del cilindro (VO) si ottiene moltiplicando il cerchio di base per l'altezza = R * R * 3.14 * H.

È abbastanza semplice passare al flow-chart e da questo al listato BASIC relativo.

```
 10 REM SUPERFICI E VOLUME DI UN CILINDRO
 15 REM PULIZIA SCHERMO E INTRODUZIONI DATI DEL CILINDRO
 20 CLS
 30 INPUT "RAGGIO = "; R
 40 INPUT "ALTEZZA = "; H
 50 REM STAMPA IL MENU
 60 PRINT "PREMI :"
 70 PRINT "1 PER SUPERFICIE LATERALE"
 80 PRINT "2 PER SUPERFICIE TOTALE"
 90 PRINT "3 PER VOLUME"
100 PRINT "4 PER NUOVI DATI"
110 PRINT "0 PER FINIRE"
120 REM SCELTA
130 INPUT "SCELGO "; SCELTA
140 IF SCELTA = 1 THEN GOTO 200
150 IF SCELTA = 2 THEN GOTO 300
160 IF SCELTA = 3 THEN GOTO 400
170 IF SCELTA = 4 THEN GOTO 20
180 IF SCELTA = 0 THEN GOTO 500
190 IF SCELTA > 4 THEN GOTO 120
200 REM CALCOLO E STAMPA SUPERFICIE LATERALE
210 LET SLAT = R * 2 * 3.14 * H
220 PRINT "SUPERFICIE LATERALE = "; SLAT
230 GOTO 120
300 REM CALCOLO E STAMPA SUPERFICIE TOTALE
310 LET SOT = R * 2 * 3.14 * H + 2 * R * R * 3.14
320 PRINT "SUPERFICIE TOTALE = "; SOT
330 GOTO 120
400 REM CALCOLO E STAMPA VOLUME
410 LET VO = R * R * 3.14 * H
420 PRINT "VOLUME = "; VO
430 GOTO 120
500 END
```

Da questi due semplici esempi emergono già alcuni aspetti fondamentali: un computer, grazie alla sua memoria e ai programmi che tu inserisci, riesce a svolgere i compiti che gli hai assegnato con molta più precisione e soprattutto velocità di quanto potrebbe fare chiunque.
Un computer, poi, può compiere delle scelte, come decidere quale tra due numeri sia quello maggiore, grazie a quell'apparato di conoscenze che tu gli dai. Un computer, inoltre, è in grado di "tirar fuori" le nozioni nel momento in cui gli servono, scegliendo ancora una volta quali dati ed informazioni deve utilizzare.

VIDEOESERCIZI

Annota nello spazio apposito il risultato da te previsto per ciascun esercizio proposto e poi verificalo con la soluzione del tuo MSX. Se avrai commesso anche un solo errore ripassa la lezione.

```
10 REM: LET N$ = "PIERO"
20 PRINT N$
```

```
10 PRINT "NON CREDO"
20 PRINT "CHE TU RIESCA"
30 CLS
40 PRINT "A VEDERE QUALCOSA"
```

```
10 CLS
20 PRINT "ADESSO SI"
```

```
10 CLS
20 REM: GOTO 40
30 PRINT "VIDEOBASIC"
40 PRINT "JACKSON"
```

```
10 LET P=0: LET S=1
20 IF P=1 THEN LET S=0
30 PRINT S, P
```

```
10 LET A = 129 : LET C = 131
20 IF A > C THEN LET C = A
30 IF C > A THEN LET A = C
40 PRINT A, C
```